Evolving Evaluation Functions for Collectible Card Game AI

Radosław Miernik and Jakub Kowalski

4 February 2021

Problem statement

Collectible Card Games (CCGs) are two-player, turn-based, multi-action games with imperfect information and randomness. (There are, of course, exceptions.)

Problem statement

Collectible Card Games (CCGs) are two-player, turn-based, multi-action games with imperfect information and randomness. (There are, of course, exceptions.)

For all of these reasons, it is infeasible to exhaustively explore the state tree, even for one's own turn – let alone the opponent's.

Problem statement

Collectible Card Games (CCGs) are two-player, turn-based, multi-action games with imperfect information and randomness. (There are, of course, exceptions.)

For all of these reasons, it is infeasible to exhaustively explore the state tree, even for one's own turn – let alone the opponent's.

Therefore, while creating an agent, it is common to rely on all kinds of heuristic functions that limit the search space. Most commonly it is a hand-crafted state evaluation, a linear combination of game-specific features, or a neural network.

Test bed

As most human-playable CCGs are extremely complex, we have decided to use Legends of Code and Magic[1] (LoCM) as our test bed. It is a CCG designed for AI research – all cards' effects are deterministic, and agents play in a fair arena mode.

[1] https://legendsofcodeandmagic.com

Idea

1. Compare representations of heuristic functions with different capabilities.

Idea

1. Compare representations of heuristic functions with different capabilities.
2. Measure how does the evolution target impact the result.

Idea

1. Compare representations of heuristic functions with different capabilities.
2. Measure how does the evolution target impact the result.
3. See whether bootstrapping a more capable model with a simpler one leads to better results than both of them alone.

Idea

1. Compare representations of heuristic functions with different capabilities.
2. Measure how does the evolution target impact the result.
3. See whether bootstrapping a more capable model with a simpler one leads to better results than both of them alone.
4. Evaluate the results in a real-world scenario – a tournament using agents from the Strategy Card Game AI Competition.

1. Representations comparison

We have implemented three representations:

- `Linear` – a linear combination of the features.

1. Representations comparison

We have implemented three representations:

- `Linear` – a linear combination of the features.
- `BinaryTree` – a binary tree with features and constants in leafs, and binary operators in nodes ($+$, $-$, $*$, *max*, *min*).

1. Representations comparison

We have implemented three representations:

- `Linear` – a linear combination of the features.
- `BinaryTree` – a binary tree with features and constants in leafs, and binary operators in nodes ($+$, $-$, $*$, *max*, *min*).
- `Tree` – a generalization of `BinaryTree` to n-ary trees. The operators are sum ($\sum$), multiplication ($\prod$), *max*, *min*, and a unary negation.

1. Representations comparison

We have implemented three representations:

- `Linear` – a linear combination of the features.
- `BinaryTree` – a binary tree with features and constants in leafs, and binary operators in nodes ($+$, $-$, $*$, *max*, *min*).
- `Tree` – a generalization of `BinaryTree` to n-ary trees. The operators are sum ($\sum$), multiplication ($\prod$), *max*, *min*, and a unary negation. To ensure that the operations are well-defined, all nodes have at least one subtree.

1. Representations comparison

There are in total 20 game-specific features. First twelve refer to the global state (six for each player): current mana, deck size, health, max mana, number of cards to draw next turn, and next rune. The rest are card-specific: attack, defense, and six flags for keywords (0.0 or 1.0).

1. Representations comparison

There are in total 20 game-specific features. First twelve refer to the global state (six for each player): current mana, deck size, health, max mana, number of cards to draw next turn, and next rune. The rest are card-specific: attack, defense, and six flags for keywords (0.0 or 1.0).

Each representation implements two operations: `evalState` (based on global state features) and `evalCard` (based on card features). Both tree-based representations store two separate trees – one with global features and one with card features.

1. Representations comparison

There are in total 20 game-specific features. First twelve refer to the global state (six for each player): current mana, deck size, health, max mana, number of cards to draw next turn, and next rune. The rest are card-specific: attack, defense, and six flags for keywords (0.0 or 1.0).

Each representation implements two operations: `evalState` (based on global state features) and `evalCard` (based on card features). Both tree-based representations store two separate trees – one with global features and one with card features.

The final state evaluation is a sum of `evalState` and `evalCard` for each own card on the board, minus `evalState` of the opponent, and `evalCard` for each of opponent's cards.

2. Evolution target

Every evolution scheme evaluates individuals either by comparing how well they deal with a specified task, without a normalized score, or by using an external, predefined goal.

2. Evolution target

Every evolution scheme evaluates individuals either by comparing how well they deal with a specified task, without a normalized score, or by using an external, predefined goal.

Both approaches have natural interpretations for CCGs – a win rate against each other and a win rate against a fixed opponent respectively.

2. Evolution target

Every evolution scheme evaluates individuals either by comparing how well they deal with a specified task, without a normalized score, or by using an external, predefined goal.

Both approaches have natural interpretations for CCGs – a win rate against each other and a win rate against a fixed opponent respectively.

Once again, we have decided to test three approaches:

- `progressive` – using a standard in-population evaluation.

2. Evolution target

Every evolution scheme evaluates individuals either by comparing how well they deal with a specified task, without a normalized score, or by using an external, predefined goal.

Both approaches have natural interpretations for CCGs – a win rate against each other and a win rate against a fixed opponent respectively.

Once again, we have decided to test three approaches:

- `progressive` – using a standard in-population evaluation.
- `weak-op` – using a baseline agent of LoCM called `Baseline2`.

2. Evolution target

Every evolution scheme evaluates individuals either by comparing how well they deal with a specified task, without a normalized score, or by using an external, predefined goal.

Both approaches have natural interpretations for CCGs – a win rate against each other and a win rate against a fixed opponent respectively.

Once again, we have decided to test three approaches:

- `progressive` – using a standard in-population evaluation.
- `weak-op` – using a baseline agent of LoCM called `Baseline2`.
- `strong-op` – using one of the best evolved agents.

3. Model bootstrapping

The tree-based models are more general but also harder to learn than a simple linear combination. The ideal scenario would be to reach a limit of optimization based on the linear representation, encode obtained solutions into the tree format, and continue evolution using this stronger model.

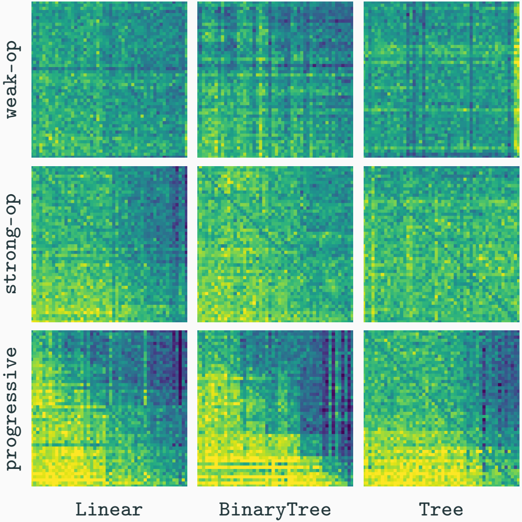
3. Model bootstrapping

The tree-based models are more general but also harder to learn than a simple linear combination. The ideal scenario would be to reach a limit of optimization based on the linear representation, encode obtained solutions into the tree format, and continue evolution using this stronger model.

We have done that by taking the evolved `Linear` agents and either continuing their evolution (`Linear-from-Linear`) or transforming them into trees and continuing the evolution as such (`Tree-from-Linear`).

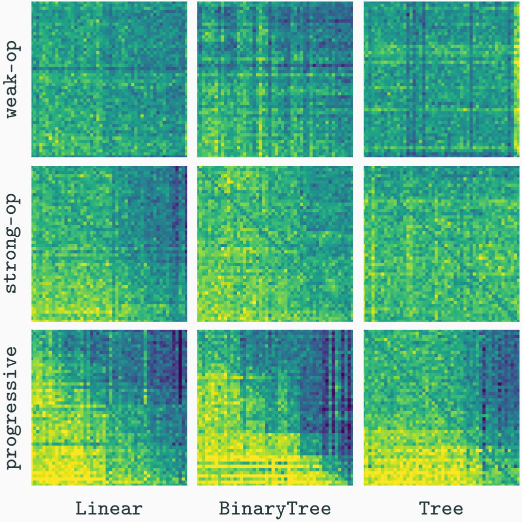
Results

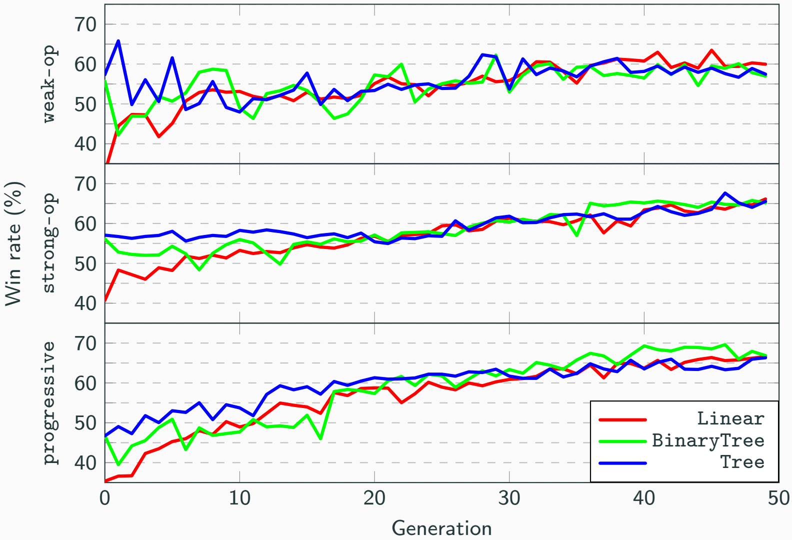
Self-play win rate heatmaps. Each cell represents how well the best individual of generation on the y-axis plays against the best individual of generation on the x-axis.



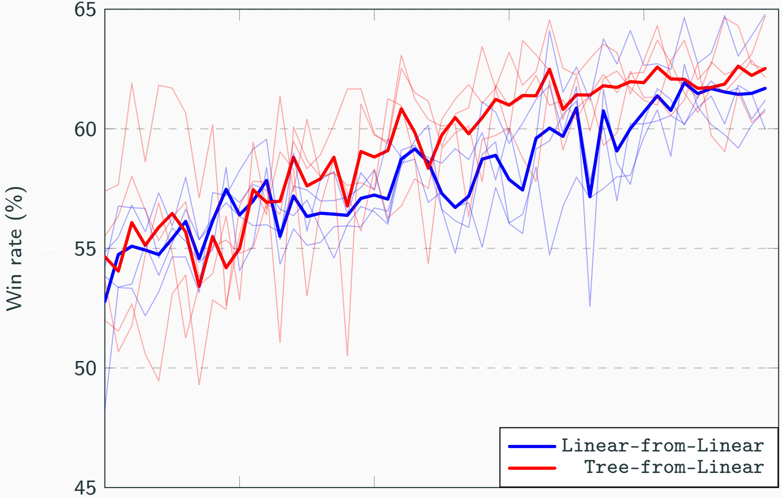
Self-play win rate heatmaps. Each cell represents how well the best individual of generation on the y-axis plays against the best individual of generation on the x-axis.

A light section on the bottom left, present in all three `progressive` agents, proves that as the evolution progresses, all individuals are increasingly better at self-play.

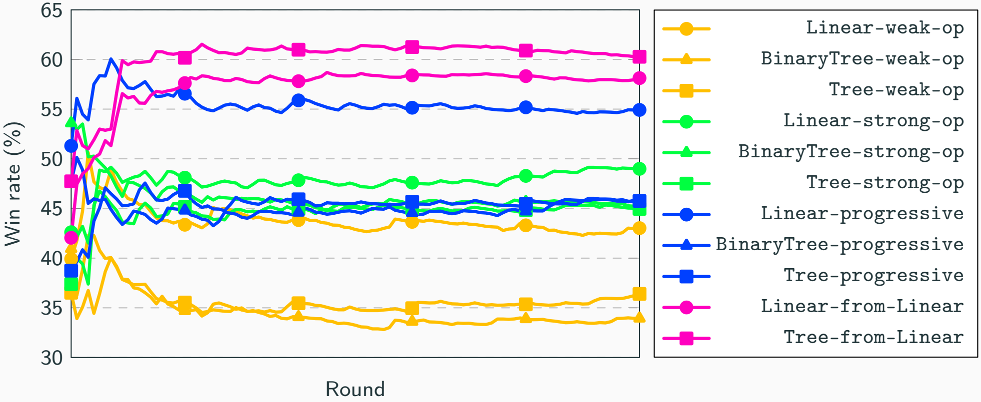




Evolution progress of all the agents. Best individuals from a generation (x-axis) fought against the top individuals of all own generations, yielding an average win rate (y-axis).



Evolution progress of the `*-from-Linear` agents. Best individuals from a generation (x-axis) fought against the top individuals of all own generations, yielding an average win rate (y-axis). The two bold lines average the thin, semi-transparent lines that are the averaged results of agents with the same base.



A subset of the tournament results. All scores (y-axis) stabilize as the number of rounds (x-axis) increases.

Thank you!